

Formalization of the Schwartz-Zippel Lemma

Sunpill Kim and Yong Kiam Tan

February 6, 2026

Abstract

This short entry formalizes a version of the Schwartz-Zippel lemma for probabilistic (multivariate) polynomial identity testing. The entry includes a textbook example using the lemma to test for perfect matchings in a bipartite graph. The lemma is attributed to several independent authors, including Schwartz [3], Zippel [4], and DeMillo and Lipton [1]; a historical perspective is given by Lipton [2].

Contents

1	The Schwartz-Zippel Lemma	1
2	A Probabilistic Test for Perfect Matchings	4

1 The Schwartz-Zippel Lemma

theory *Schwartz-Zippel* **imports**

Factor-Algebraic-Polynomial.Poly-Connection

Polynomials.MPoly-Type

HOL-Probability.Product-PMF

begin

This theory formalizes the Schwartz-Zippel lemma for multivariate polynomials (*mpoly*).

lemma *schwartz-zippel-uni*:

fixes $P :: ('a::\text{idom}) \text{Polynomial.poly}$

fixes $S :: 'a \text{ set}$

assumes $\text{finite } S \ S \neq \{\}$

assumes $\text{Polynomial.degree } P \leq d$

assumes $P \neq 0$

shows $\text{measure-pmf.prob (pmf-of-set } S) \{r. \text{poly } P \ r = 0\} \leq \text{real } d / \text{card } S$
(*proof*)

lemma *degree-mpoly-to-poly* [*simp*]:

assumes $\text{vars } p = \{x\}$
shows $\text{Polynomial.degree } (\text{mpoly-to-poly } x \ p) = \text{MPoly-Type.degree } p \ x$
 ⟨proof⟩

lemma *total-degree-add*: $\text{total-degree } (x + y) \leq \max (\text{total-degree } x) (\text{total-degree } y)$
 ⟨proof⟩

lemma *total-degree-sum*:
assumes *finite S*
shows $\text{total-degree } (\text{sum } f \ S) \leq \text{Max } ((\text{total-degree } \circ f) \ ` \ S)$
 ⟨proof⟩

lemma *coeff-mpoly-to-mpoly-poly-restrict*:
shows $\text{coeff } (\text{mpoly-to-mpoly-poly } x \ P) \ i = \text{sum } (\lambda m. \text{MPoly-Type.monom } (\text{remove-key } x \ m) (\text{MPoly-Type.coeff } P \ m) \ \text{when } \text{lookup } m \ x = i) (\text{Poly-Mapping.keys } (\text{mapping-of } P))$
 ⟨proof⟩

lemma *Max-le-Max*:
assumes $A \neq \{\}$
assumes *finite A finite B*
assumes $\bigwedge a. a \in A \implies \exists b. b \in B \wedge a \leq b$
shows $\text{Max } A \leq \text{Max } B$
 ⟨proof⟩

lemma *sum-lookup-remove-key*:
 $\text{sum } (\text{lookup } (\text{remove-key } x \ y)) (\text{keys } (\text{remove-key } x \ y)) + \text{lookup } y \ x = \text{sum } (\text{lookup } y) (\text{keys } y)$
 ⟨proof⟩

lemma *total-degree-nonzero*:
assumes $P \neq 0$
shows $\text{total-degree } P = \text{Max } ((\lambda x. \text{sum } (\text{lookup } x) (\text{keys } x)) \ ` \ \text{keys } (\text{mapping-of } P))$
 ⟨proof⟩

lemma *poly-mapping-eq-iff*: $(m = m') \longleftrightarrow (\forall i. \text{lookup } m \ i = \text{lookup } m' \ i)$
 ⟨proof⟩

lemma *total-degree-coeff-mpoly-to-mpoly-poly*:
assumes $\text{coeff } (\text{mpoly-to-mpoly-poly } x \ P) \ i \neq 0$

shows $\text{total-degree } (\text{coeff } (\text{mpoly-to-mpoly-poly } x P) i) + i \leq \text{total-degree } P$
<proof>

lemma *degree-le-total-degree*:

shows $\text{MPoly-Type.degree } P x \leq \text{total-degree } P$
<proof>

lemma *insertion-update*:

shows $\text{insertion } (f(x := r)) P = \text{poly } (\text{map-poly } (\text{insertion } f) (\text{mpoly-to-mpoly-poly } x P)) r$
<proof>

lemma *measure-pmf-prob-dependent-product-bound*:

assumes $\text{countable } A \wedge i. \text{countable } (B i)$
assumes $\bigwedge a. a \in A \implies \text{measure-pmf.prob } N (B a) \leq r$
shows $\text{measure-pmf.prob } (\text{pair-pmf } M N) (\text{Sigma } A B) \leq \text{measure-pmf.prob } M A * r$
<proof>

lemma *measure-pmf-prob-dependent-product-bound'*:

assumes $\text{countable } (A \cap \text{set-pmf } M) \wedge i. \text{countable } (B i \cap \text{set-pmf } N)$
assumes $\bigwedge a. a \in A \cap \text{set-pmf } M \implies \text{measure-pmf.prob } N (B a \cap \text{set-pmf } N) \leq r$
shows $\text{measure-pmf.prob } (\text{pair-pmf } M N) (\text{Sigma } A B) \leq \text{measure-pmf.prob } M A * r$
<proof>

lemma *finite-set-pmf-Pi-pmf*:

assumes $\text{finite } A$
assumes $\bigwedge x. x \in A \implies \text{finite } (\text{set-pmf } (p x))$
shows $\text{finite } (\text{set-pmf } (\text{Pi-pmf } A \text{ def } p))$
<proof>

theorem *schwartz-zippel*:

fixes $P :: ('a::\text{idom}) \text{mpoly}$
fixes $S :: 'a \text{set}$
assumes $S: \text{finite } S \ S \neq \{\}$
assumes $V: \text{finite } V$
assumes $P: \text{total-degree } P \leq d \ P \neq 0 \ \text{vars } P \subseteq V$
shows $\text{measure-pmf.prob } (\text{Pi-pmf } V 0 (\lambda i. \text{pmf-of-set } S)) \{f. \text{insertion } f P = 0\} \leq \text{real } d / \text{card } S$
<proof>

end

2 A Probabilistic Test for Perfect Matchings

```

theory Rand-Perfect-Matching imports
  Schwartz-Zippel
  Jordan-Normal-Form.Determinant
begin

```

We use a simple representation of bipartite graphs (with same no. vertices) $V::nat$, $E::(nat \times nat)$ list where V is the number of vertices in each partition and $(x,y) \in E$ represents an edge between vertex x in the left partition and vertex y in the right partition.

```

definition is-matching::
  (nat × nat) set ⇒ (nat × nat) set ⇒ bool
where is-matching E match ⇔
  match ⊆ E ∧
  inj-on fst match ∧
  inj-on snd match

definition has-perfect-matching::
  nat ⇒ (nat × nat) set ⇒ bool
where has-perfect-matching V E ⇔
  (∃ match. is-matching E match ∧ card match = V)

```

```

definition adj-mat::nat ⇒ (nat × nat) set ⇒
  int mpoly mat
where adj-mat V E =
  mat V V (λ(i,j).
    if (i,j) ∈ E then Var (i*V+j) else 0)

```

```

lemma adj-mat-square[simp]:
shows
  dim-row (adj-mat V E) = V
  dim-col (adj-mat V E) = V
⟨proof⟩

```

```

lemma perfect-match-set-map-fst:
assumes E ⊆ {0..<V} × {0..<V}
assumes is-matching E match
assumes card match = V
shows fst ` match = {0..<V}
⟨proof⟩

```

```

lemma perfect-match-set-map-snd:
assumes E ⊆ {0..<V} × {0..<V}
assumes is-matching E match
assumes card match = V
shows snd ` match = {0..<V}
⟨proof⟩

```

lemma *is-matching-permutes*:
assumes $E \subseteq \{0..<V\} \times \{0..<V\}$
assumes *is-matching* E *match*
assumes *card match* = V
obtains f **where**
 f *permutes* $\{0..<V\}$
 $\bigwedge i. i < V \implies (i, f i) \in E$
 \langle *proof* \rangle

lemma *Var-not-0*:
shows $\text{Var } x \neq (0::'a::\text{idom } \text{mpoly})$
 \langle *proof* \rangle

lemma *sum-monom-0-iff*:
assumes *fin*: *finite* S
and $g: \bigwedge i j. i \in S \implies j \in S \implies g i = g j \implies i = j$
shows $\text{sum } (\lambda i. \text{MPoly-Type.monom } (g i) (f i)) S = 0 \longleftrightarrow (\forall i \in S. f i = 0)$
 \langle *is ?l = ?r* \rangle
 \langle *proof* \rangle

lemma *prod-Var*:
assumes *finite* S
shows $\text{prod } (\lambda i. \text{Var } (f i)) S =$
 $\text{MPoly-Type.monom } (\text{sum } (\lambda i. \text{monomial } 1 (f i)) S) 1$
 \langle *proof* \rangle

lemma *det-adj-mat*:
shows $\text{det } (\text{adj-mat } V E) =$
 $(\sum p \mid p \text{ permutes } \{0..<V\}).$
 $\text{MPoly-Type.monom } ($
 $\text{sum } (\lambda i.$
 $\text{monomial } 1 (i * V + p i)) \{0..<V\})$
 $(\text{if } \forall i < V. (i, p i) \in E \text{ then}$
 $\text{of-int } (\text{sign } p)$
 $\text{else } 0))$
 \langle *proof* \rangle

lemma *vars-prod-Var*:
assumes *finite* S
shows $\text{vars } (\text{prod } \text{Var } S) = S$
 \langle *proof* \rangle

lemma *prod-Var-eq*:
assumes *finite* S *finite* T
assumes $\text{prod } \text{Var } S = \text{prod } \text{Var } T$
shows $S = T$
 \langle *proof* \rangle

lemma *pair-enc-eq*:

assumes $a * V + b = c * V + d$

assumes $b < V$ $d < V$

shows $b = (d::nat)$

<proof>

lemma *sum-monomial-eq*:

assumes f *permutes* $\{0..<V\}$

assumes g *permutes* $\{0..<V\}$

assumes

$(\sum i = 0..<V.$

$monomial (1::nat) (i * V + (f i::nat))) =$

$(\sum i = 0..<V.$

$monomial (1::nat) (i * V + g i))$

shows $f = g$

<proof>

lemma *perfect-matching-det*:

assumes $E \subseteq \{0..<V\} \times \{0..<V\}$

assumes *is-matching* E *match*

assumes $card\ match = V$

shows $det (adj\text{-}mat\ V\ E) \neq 0$

<proof>

lemma *det-perfect-matching*:

assumes $E \subseteq \{0..<V\} \times \{0..<V\}$

assumes $det (adj\text{-}mat\ V\ E) \neq 0$

obtains *match* **where**

is-matching E *match*

$card\ match = V$

<proof>

lemma *has-perfect-matching-iff*:

assumes $E \subseteq \{0..<V\} \times \{0..<V\}$

shows $has\text{-}perfect\text{-}matching\ V\ E \iff det (adj\text{-}mat\ V\ E) \neq 0$

<proof>

lemma *sum-when-1*:

assumes *finite* S $x \in S$

shows $(\sum xa \in S. 1\ when\ xa = x) = 1$

<proof>

lemma *total-degree-monom*:

assumes *finite* S

shows $total\text{-}degree\ (MPoly\text{-}Type.monom\ (sum\ (\lambda i. monomial\ (Suc\ 0)\ i)\ S)\ c) =$
 $(if\ c = 0\ then\ 0\ else\ card\ S)$

<proof>

lemma *total-degree-geI*:

assumes $m \in \text{keys } (\text{mapping-of } p) \ (\sum v \in \text{keys } m. \text{lookup } m \ v) \geq n$

shows $\text{total-degree } p \geq n$

<proof>

lemma *total-degree-0-iff*: $\text{total-degree } p = 0 \longleftrightarrow \text{vars } p = \{\}$

<proof>

lemma *total-degree-0E*: $\text{total-degree } p = 0 \implies (\bigwedge c. p = \text{Const } c \implies P) \implies P$

<proof>

lemma *total-degree-ex*:

assumes $p \neq 0$

shows $\exists m. m \in \text{keys } (\text{mapping-of } p) \wedge (\sum v \in \text{keys } m. \text{lookup } m \ v) = \text{total-degree } p$

<proof>

lemma *coeff-times-const-left [simp]*: $\text{MPoly-Type.coeff } (\text{Const } c * p) \ m = c * \text{MPoly-Type.coeff } p \ m$

<proof>

lemma *total-degree-times-const-left*: $\text{total-degree } (\text{Const } c * p) \leq \text{total-degree } p$

<proof>

lemma *total-degree-of-Const [simp]*: $\text{total-degree } (\text{Const } x) = 0$

<proof>

lemma *total-degree-of-int [simp]*: $\text{total-degree } (\text{of-int } x) = 0$

<proof>

lemma *total-degree-det-adj-mat*: $\text{total-degree } (\text{det } (\text{adj-mat } V \ E)) \leq V$

<proof>

lemma *arith*:

assumes $i < V$

assumes $j < (V::\text{nat})$

shows $i * V + j < V^2$

<proof>

lemma *vars-det-adj-mat*:

shows $\text{vars } (\text{det } (\text{adj-mat } V \ E)) \subseteq \{0..<V^2\}$

<proof>

definition *int-adj-mat*::

$(\text{nat} \Rightarrow \text{int}) \Rightarrow$

$\text{nat} \Rightarrow$

$(\text{nat} \times \text{nat}) \text{ set} \Rightarrow$

int mat

where $\text{int-adj-mat } f \ V \ E =$

$mat\ V\ V\ (\lambda(i,j).$
 $\text{if } (i,j) \in E \text{ then } f\ (i* V + j) \text{ else } 0)$

lemma *map-mat-prod-def*:
shows $map\ mat\ f\ A \equiv$
 $Matrix.mat\ (dim\ row\ A)\ (dim\ col\ A)$
 $(\lambda(i,j). f\ (A\ \$\$ (i,j)))$
 $\langle proof \rangle$

lemma *int-adj-mat*:
shows $int\ adj\ mat\ f\ V\ E =$
 $map\ mat\ (insertion\ f)\ (adj\ mat\ V\ E)$
 $\langle proof \rangle$

lemma *det-int-adj-mat*:
shows $det\ (int\ adj\ mat\ f\ V\ E) =$
 $insertion\ f\ (det\ (adj\ mat\ V\ E))$
 $\langle proof \rangle$

definition *test-perfect-matching* :: $int \Rightarrow nat \Rightarrow (nat \times nat)\ set \Rightarrow bool\ pmf$
where $test\ perfect\ matching\ n\ V\ E =$
 $do\ \{$
 $f \leftarrow Pi\ pmf\ (\{0..<V^2\})\ 0\ (\lambda i. pmf\ of\ set\ \{0::int..<n\});$
 $return\ pmf\ (det\ (int\ adj\ mat\ f\ V\ E) \neq 0)$
 $\}$

theorem *test-perfect-matching-false-positive*:
assumes $E \subseteq \{0..<V\} \times \{0..<V\}$
assumes $\neg has\ perfect\ matching\ V\ E$
shows $pmf\ (test\ perfect\ matching\ n\ V\ E)\ True = 0$
 $\langle proof \rangle$

lemma *test-perfect-matching-true-negative*:
assumes $E \subseteq \{0..<V\} \times \{0..<V\}$
assumes $\neg has\ perfect\ matching\ V\ E$
shows $pmf\ (test\ perfect\ matching\ n\ V\ E)\ False = 1$
 $\langle proof \rangle$

theorem *test-perfect-matching-false-negative*:
assumes $(n::nat) > 0$
assumes $E \subseteq \{0..<V\} \times \{0..<V\}$
assumes $has\ perfect\ matching\ V\ E$
shows $pmf\ (test\ perfect\ matching\ n\ V\ E)\ False \leq V / n$
 $\langle proof \rangle$

end

References

- [1] R. A. DeMillo and R. J. Lipton. A probabilistic remark on algebraic program testing. *Inf. Process. Lett.*, 7(4):193–195, 1978.
- [2] R. Lipton. The curious history of the Schwartz-Zippel lemma, 2009. Accessed on Apr 21, 2023.
- [3] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, 1980.
- [4] R. Zippel. Probabilistic algorithms for sparse polynomials. In E. W. Ng, editor, *EUROSAM*, volume 72 of *LNCS*, pages 216–226. Springer.